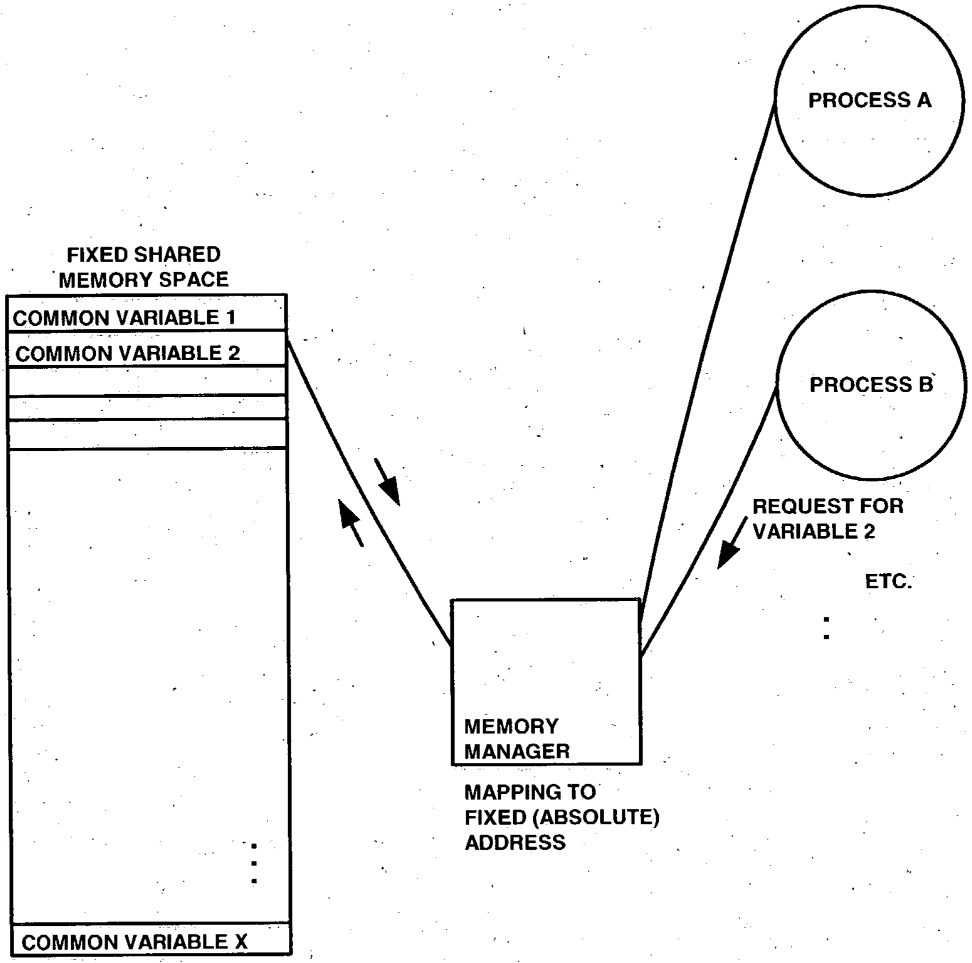

(PRIOR ART)

FIG. 1

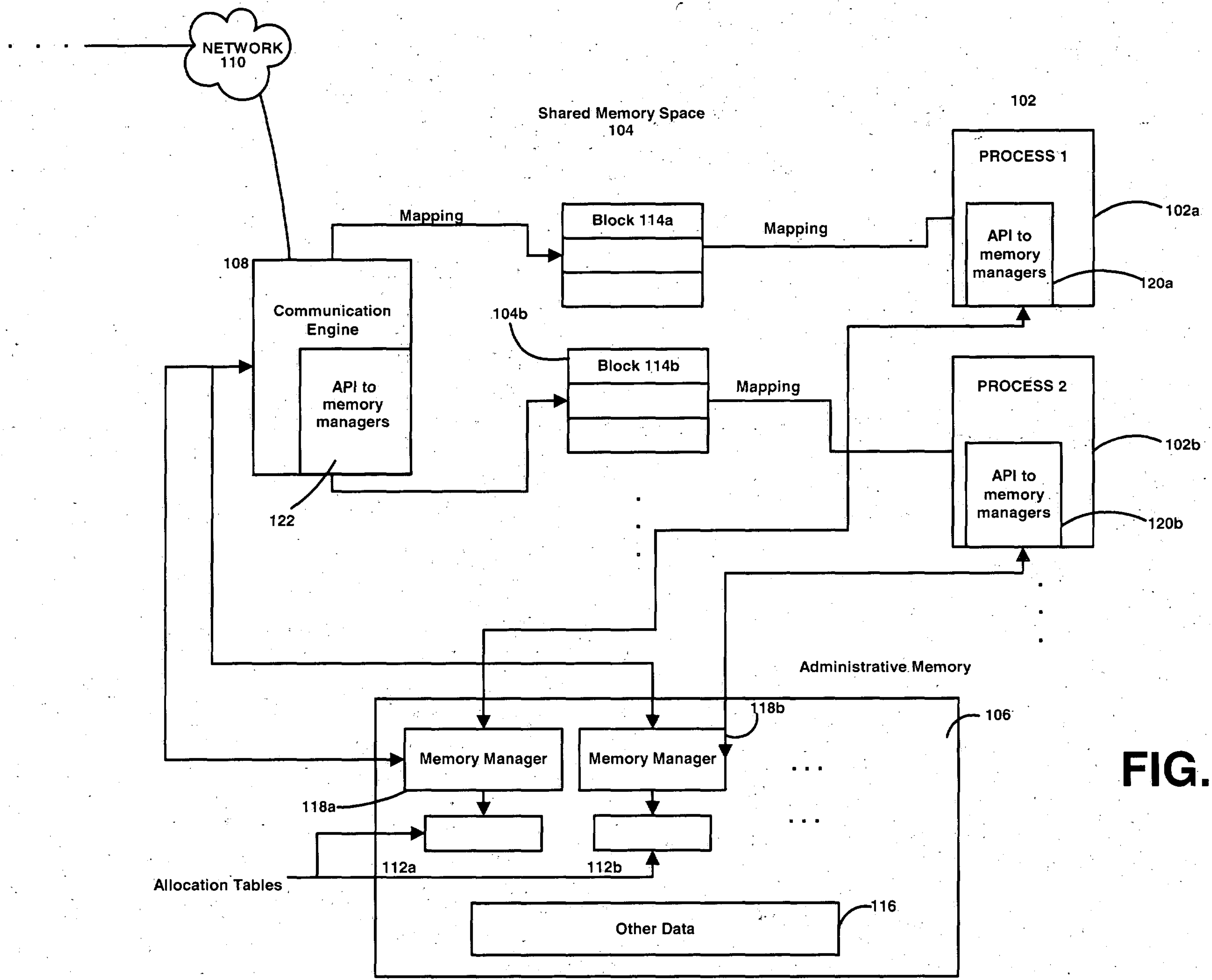
NETWORK 110
Shared Memory Space 104
102
PROCESS 1
Mapping
Block 114a
Mapping
102a
API to memory managers
120a
108
Communication Engine
API to memory managers
104b
Block 114b
Mapping
PROCESS 2
102b
API to memory managers
120b
122
Administrative Memory
106
118b
Memory Manager
Memory Manager
118a
112a
112b
Allocation Tables
Other Data
116

FIG. 2

```
//this code registers the application using a unique identifying value and
//supplying a callback structure.  Underneath the api creates the communication
//infrastructure and shared memory for caching
// this may only be done once within the application

Dc.RegisterSMPQ(bAppReg, //app id
        &MessageCallBackHandler //callback functions
);

// this code creates a session with more or more remote computers.  It sets the priority of the
//session, a debug string, timeouts and other various parameters
//a session can be created with N computers

hr = Dc.CreateSession(&pSess, //out param that will receive the session object
                eSESSION_NORMAL_PRI, //session weighted priority
                FALSE,  //have the infrastructure periodically check the session liveness
                L"3a047c3f-49da-4aa2-9ec9-a8dda18bc2b4",
                2500,   //timeout for the session creation
                dwSReg, //
                dcRemoteComputerNameList, //list of computers, and their quotas
                1, //number of computer names in the RemoteComputerNameList
                &vcResponse //what happened when trying to create the session
);


//creates a message on a specific session

Dc.CreateMessage(reinterpret_cast<IMessage**>(&pMsg), pSess);


//set the data on the message

pMsg->SetData(pbData, dwDataSize);

//send the message; specifying timeout, unicast or multicast, synchronous or async
//One or more response will be received on the callback function if ASYNCHRONOUS
//is supplied during registration; otherwise the call blocks  until a response is received

Dc.Send(pMsg, 250, unicast, SYNCHRONOUS, pResponseMsg);

delete pMsg;

//the message object can be recycled if the application wishes
//Sending a file (whether 1 byte or N gigabytes) is straightforward, chunking and multiplexing
the //data may be handled by the messaging system:
//this line of code registers the application using their unique identifying value and
//supplying a callback structure.  Underneath the api creates the communication
//infrastructure and shared memory for caching
// this may only be done once within the application

Dc.RegisterSMPQ(bAppReg, //app id
        &MessageCallBackHandler //callback functions
);

// this code creates a session with more or more remote computers.  It sets the priority of the
//session, a debug string, timeouts and other various params
//a session can be created with N computers

hr = Dc.CreateSession(&pSess, //out param that will receive the session object
                eSESSION_NORMAL_PRI, //session weighted priority
                FALSE,  //have the infrastructure periodically check the session liveness
                L"3a047c3f-49da-4aa2-9ec9-a8dda18bc2b4",
                2500,   //timeout for the session creation
                dwSReg, //
                dcRemoteComputerNameList, //list of computers, and their quotas
                1, //number of computer names in the RemoteComputerNameList
                &vcResponse //what happened when trying to create the session
);


//Create a file message instead of a normal message

Dc.CreateFileMessage(&pFileMsg,
                pSess,
                TRUE);


// build the file name of the source file

wcscpy(szFileName, L"c:\\TransferFrom\\trans");
wcscat(szFileName, _itow(i, szBuffer, 10));
wcscat(szFileName, L".txt");


//set the file name

pFileMsg->SetLocalPathAndFilename(szFileName);


//build the file name for the destination path (on a remote box)

wcscpy(szFileName, L"c:\\Transferto\\file");
wcscat(szFileName, _itow(i, szBuffer, 10));


//set  file name data (application can define and use any format it wishes)
//to send data pertinent for recreation on the remote process

pFileMsg->SetData(szFileName, (wcslen(szFileName) * 2), FALSE, FALSE);


//at this call the system will call back into the application asking
// for fixed sized chunks of data, using the messaging system
// to cache the data to go out through the communication channel.

Dc.Send(pFileMsg, INFINITE, unicast, SYNCHRONOUS, pResponseMsg);

delete pFileMsg;
```

FIG. 3

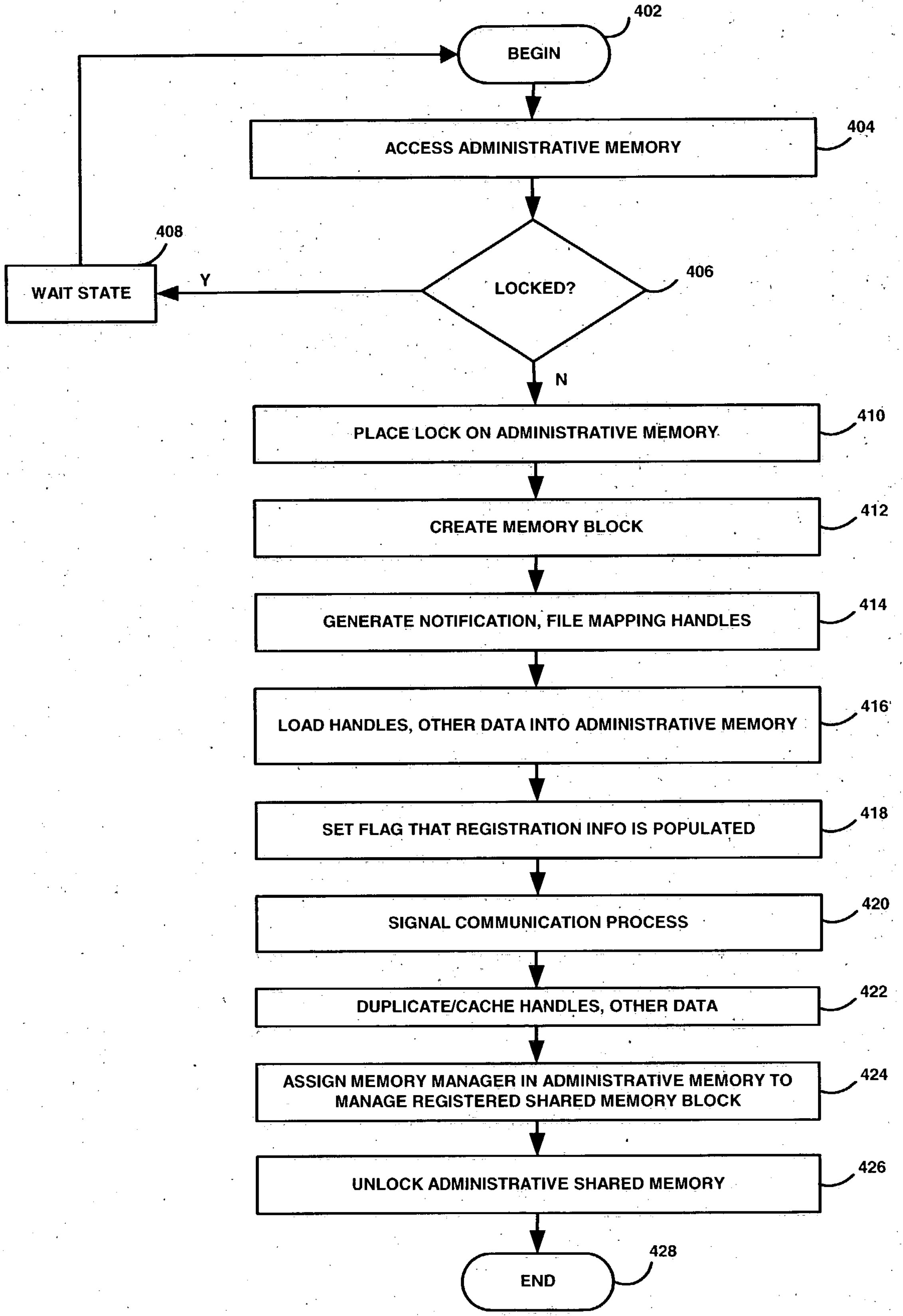
402
BEGIN
404
ACCESS ADMINISTRATIVE MEMORY
408
WAIT STATE
Y
LOCKED?
406
N
410
PLACE LOCK ON ADMINISTRATIVE MEMORY
412
CREATE MEMORY BLOCK
414
GENERATE NOTIFICATION, FILE MAPPING HANDLES
416
LOAD HANDLES, OTHER DATA INTO ADMINISTRATIVE MEMORY
418
SET FLAG THAT REGISTRATION INFO IS POPULATED
420
SIGNAL COMMUNICATION PROCESS
422
DUPLICATE/CACHE HANDLES, OTHER DATA
424
ASSIGN MEMORY MANAGER IN ADMINISTRATIVE MEMORY TO MANAGE REGISTERED SHARED MEMORY BLOCK
426
UNLOCK ADMINISTRATIVE SHARED MEMORY
428
END

FIG. 4

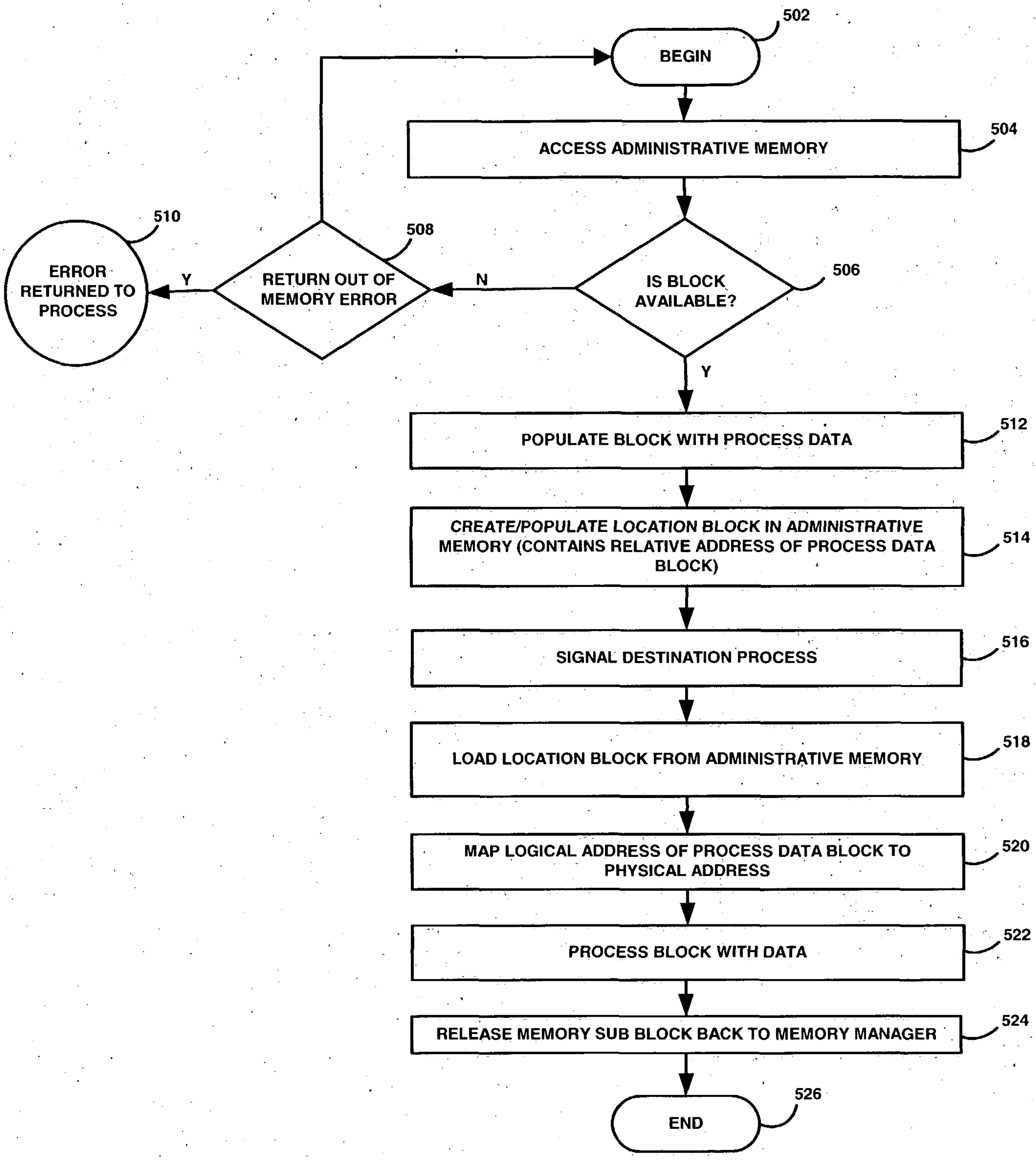
502
BEGIN
ACCESS ADMINISTRATIVE MEMORY
504
IS BLOCK AVAILABLE?
506
N
RETURN OUT OF MEMORY ERROR
508
Y
ERROR RETURNED TO PROCESS
510
Y
POPULATE BLOCK WITH PROCESS DATA
512
CREATE/POPULATE LOCATION BLOCK IN ADMINISTRATIVE MEMORY (CONTAINS RELATIVE ADDRESS OF PROCESS DATA BLOCK)
514
SIGNAL DESTINATION PROCESS
516
LOAD LOCATION BLOCK FROM ADMINISTRATIVE MEMORY
518
MAP LOGICAL ADDRESS OF PROCESS DATA BLOCK TO PHYSICAL ADDRESS
520
PROCESS BLOCK WITH DATA
522
RELEASE MEMORY SUB BLOCK BACK TO MEMORY MANAGER
524
END
526

FIG. 5